BETA BASIC NEWSLETTER No 8

****

Welcome to the eighth issue! The previous issue seems to have been fairly misprint-free; the only one reported was a minor one in PROC splay on page 6. The assignment to bb should be: LET bb="cCdDe.. etc. Thanks to all the subscribers who wrote to say that they enjoy the Newsletter, and to those who sent in contributions. If you haven't been "published" yet - better luck next time!

****

BETA BASIC 4.0 AND PRINTERS

I noticed at the last Microfair that some owners or prospective owners of BB 4.0 did not realise that they could use the program with the Kempston and Euroelectronics parallel interfaces, or with the ZX printer, without the normal problems 128k mode causes. Details are in the back of the BB 4.0 manual.

BB 4.0 also allows most Alphacom printers to be used, but in some cases merely having the printer attached causes a crash in 128k, mode, before Beta Basic can even be loaded. This is a hardware problem I can do nothing about, unfortunately. I have no idea why some Spectrum/Alphacom combinations should crash in this way, while others behave perfectly.

One further machine to cause a problem is Dixon`s 8056 serial printer. Most serial printers can COPY directly via the 128K Spectrum's serial port, using a routine built into the ROM, but the 8056 apparently does not recognise EPSON control codes, and so needs tape-loaded CODE to do a screen copy. This unfortunately is not relocatable, and the location used clashes with Beta Basic. In the past, I have relocated printer driving code for perhaps 5 or 6 printer interfaces, but frankly I'm tired of it! It takes quite a while, since I usually don't have one of the interfaces. I cannot test my work easily and therefore have to be very careful. I think a software house writing a small program (like a printer driver) that should work with lots of other programs, should make it relocatable. (In case you are wondering, Beta Basic is perhaps 50 times as large and much harder to make relocatable.) Anyway, for the fist time I baulked at doing a relocation job, and reader A. Landaw (London) has therefore had to create screens with BB 4.0, SAVE them by e.g. SAVE "picture" SCREENS, and then copy them to the printer as follows:

1. Reset the computer to normal 128K mode.
2. LOAD the 8056 printer driving software.
3. Enter FORMAT "p";1200 to set the baud rate correctly.
4. Press EDIT and select SCREEN on the menu to keep typed input in the bottom 2 limes of the screen.
5. Now LOAD e.g. "picture" SCREEN$.
6. Enter RANDOMIZE USR 63696 and the screen copy should begin.

****
UPPER AND LOWER CASE CONTROL

These simple procedures from Stephen Folly (Harmondsworth, Middx.) should be useful to those who, like myself, can never remember which type of SHIFT$ forces which case.

```
     10   LET A$="Testing"
          PRINT a$
     20   VCASE a$
          PRINT a$
     30   LCASE a$
          PRINT a$

     100  DEF PROC VCASE REF X$
             LET X$=SHIFT$(1,X$)
          END PROC

     110  DEF PROC LCASE REF X$
             LET X$=SHIFT$(2,X$)
          END PROC
```

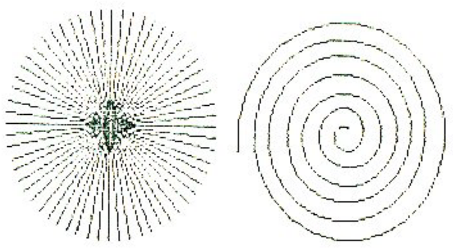
****
PROC FD - drawing at an angle.

Stephen Folly's second contribution draws a line DIS pixels long from the current position at an angle ANG (in degrees). I've added a simple demo routine (as I usually do.)

```
     10   FOR A=0 TO 359 STEP 6
             PLOT 62,88
             FD 60,A
          NEXT A

     20   LET D=16,A=0
          PLOT 135,75
          DO
             FD D,A
             LET A=A+15,D=D-.1
          LOOP UNTIL D<=0

     200  DEF PROC FD DIS,ANG
             LET ANG=ANG/180*PI
             DRAW DIS*SINE(ANG),DIS*COSE(ANG)
          END PROC
```

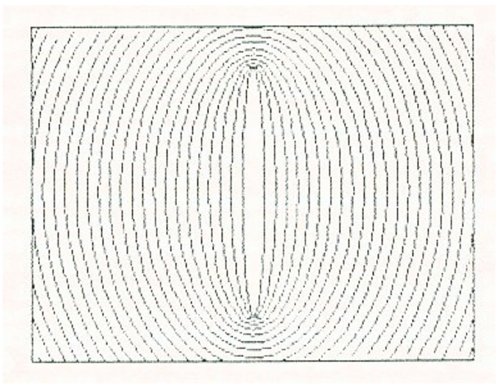


****

DRAWING CURVES WHICH ARE PARTLY OFF-SCREEN

This is another interesting submission from Ettrick Thomson (Aldeburgh, Suffolk). Here is his explanation:

"Sometimes a curve that runs off the screen can, if the program handles the display correctly, come back to the screen later on. The illustration shows a family of curves (they happen to be ellipses), the inner ones completely on the screen, the outers having off-screen portions. The program below produced all these curves. The variable s defines a particular curve, and the a-LOOP generates 100 points on it. Normally, the first of these is plotted and the rest drawn to. But if a curve runs off the screen, when it returns we must plot a point before returning to the DRAW TO. Line 80 does this under the control of flag f, normally 1, but set to zero by line 10 when we run off the screen."



"Since PLOT m,n (with XRG etc. having their default values) is actually PLOT ABS m,ABS n, line 80 must include a check that we are not in the ambiguous region: thus, if it were not for the check, the values x=-2.4,y=0 would lead to a point being plotted at x=-1.6,y=0. (Because the x coordinate of the left-hand side has been set to -2.0, so -2.4 is off-screen by -0.4, but gets "reflected" back by +0.4 - giving a point at -1.6. Ed.) On the other hand, values x=2.4; y=0 pass the check but lead to error 11, so that no point is plotted and f becomes zero again. Flag f, zeroed at line 40, also arranges that the very first point on each curve is plotted."

"This scheme is not perfect: when we return to the screen, the curve does not begin at the screen edge, but at one of the x,y points defined by the a-LOOP. This can be seen towards the right-hand end of the lower edge. (I also put a frame round the illustration to show up the screen edge. Ed.) The essential part of the scheme is lines 10, 80, and the LET f=0 of line 40. They can be added to any curve-drawing routine."

```
     10   ON ERROR
          IF error=11 THEN
             LET f=0
             RETURN
          ELSE POP
             CONTINUE
     20   LET xrg=4,xos=2,yrg=2.75,yos=1.375
     30   FOR s=.1 TO 2.4 STEP .1
     40      LET t=SQR (s*s+1),f=0
     50      FOR a=0 TO 100
     60         LET r=a*PI/50
     70         LET x=s*COSE(r),y=t*SINE(r)
     80         If f THEN
                   DRAW TO x,y
                ELSE
                   IF x> -xos AND y> -yos THEN
                      LET f=1
                      PLOT x,y
     90      NEXT a
     100  NEXT s
```

****
PROC BEEPER - a BEEP that doesn't make the CLOCK lose time.

Robert Dickson (Blackheath, London) submitted a number of interesting procedures, from which I have chosen this one which can replace the Spectrum BEEP command. Robert writes:

"Whenever the Spectrum emits a beep, the internal clock and the Beta Basic clock both stop for the duration of the beep. When this subroutine is called, the same beep is emitted, but the duration is added to the variable XYZ. If XYZ is greater than 1 then the procedure will add the integer of XYZ to the time on the Beta Basic clock, hence the beep seems not to stop the clock. N.B. Since the procedure is called via the same parameters as the normal Spectrum BEEP command, your programs may be altered by the following command:

```
     ALTER " BEEP " TO "Beeper "
```

(BEEP must be entered as a keyword.) The variable XYZ must not be used by the main program. The internal clock remains unaltered, although the procedure may be easily adapted to change the internal clock as well as, or instead of, the Beta Basic clock"

I added lines 10 to 50 as a test and fiddled with the procedure slightly. I found that time-keeping was slightly imperfect until I added line 9010, which uses my unfair advantage of having written Beta Basic! I think what was happening was that sometimes, after LET a$=TIME$() in line 9020 and before the clock time was reset at line 9040, the clock "ticked" so that the clock was being "advanced" to the time it actually already held. Line 9010 gets round this by looking at location 61186, which "ticks" 50 times a second. Its contents start at 50, change to 49, 48.... 2,1 and at zero the seconds value held by CLOCK changes. The DO-loop waits till we have at least 8/50ths. of a second to reset the clock - a time which I found was adequate by trial and error.

```
     10   CLOCK 1
          CLOCK "0000”
     20   DO
     30      beeper RND*2,1
     40      PAUSE 50
     50   LOOP

     9000 DEF PROC Beeper d,p
             LOCAL a,a$,hrs,mins,secs
             DEFAULT xyz=0
             LET xyz=xyz+d
             BEEP d,p
             IF xyz>1 THEN
                LET a=INT xyz
             ELSE
                GO TO 9060
     9010    DO
             LOOP UNTIL PEEK 61186>8
     9020    LET a$=TIME$(),hrs=VAL a$(1  TO 2), mins=VAL a$(4 TO 5),
             secs=VAL a$(7 TO 8)+a
     9030    IF secs>59 THEN
                LET secs=secs-60,mins=mins+1
                IF mins>59 THEN
                   LET mins=0,hrs=hrs+1
                   IF hrs>23 THEN
                      LET hrs=0
     9040    CLOCK USINGS("00",hrs)+USING$("00",mins)+USING$("00",s
             ecs)
     9050    LET xyz=xyz-a
     9060 END PROC
```

****
CONTROLLING "REPORTS"

The next three procedures were sent in a long while ago by Neil Exley (Bolton-le-Sands, Lanes.) prompted by procedures in issue 5. They really should have appeared in issue 6, but unfortunately I mislaid them. Better late than never!

Both of the first two procedures Neil sent use an inelegant but effective method of clearing the stack before forcing the program to halt. This is fully explained in issue 5, and I won't repeat it here. PROC END3 is a minor improvement to PROC END2 (see issue 5) with a different line 120. Neil says:

"This will alter the messy "0 OK, 120:1" message (which is showing the current line and statement numbers) to a neat "0 OK, 0:1". In fact you can matte it print any line and statement numbers by changing the DPOKE and POKE values."

```
     100  DEF PPOC end3
     110     IF DPEEK(23613)+3<DPEEK(23730) THEN
                GO SUB 130
                POP
                GO TO 110
     120     DPOKE 22621,0
             POKE 23623,0
             GO TO 10000
     110  END PROC
```

"A variation of 'Printing error messages' (given in issue 5) is the procedure below, called 'report'. This will force any error to be created. The first parameter is the value of the error to be produced, and the second and third parameters are the line and statement numbers given with the error."

I confess I couldn't understand some features of this PROC and tried to alter it. When it stopped working I gradually worked out why it *has* to be written the way it is. So - don't meddle, and don't worry if it looks odd! (As an aside - anyone unfortunate enough to have the awful manual issued with the Spectrum + will be very confused by the POKES and PEEKS of Spectrum system variables that are often part of the more esoteric programs in this Newsletter, because information on these variables was no longer included in that manual. Subscriber P.A. Basheer (Rig Dhabi-2, Abu Dhabi) has kindly informed me that the Spectrum +2 manual is available from Amstrad for £10.50+P&P. This manual is much better, and has all the old "techie" stuff as well as new information on the 128K(mode features.)

```
     200  DEF PROC report v,L,s
             DEFAULT L=0,s=1
             LET lin=L,stm=s-1,cde=v-1
     210     IF DPEEK (23623)+3 < DPEEK (23730)THEN GO SUB 230
                POP
                GO TO 210
     220     IF cde<>-1 THEN
                KEYIN " DPOKE 23621,lin: POKE 23623,stm: POKE 23610,
                cde"
             ELSE DPOKE 27621,lin
                POKE 23627,,stm
                GO TO 1000
          END PROC
```

****
FROG LINES - copying a program line into a string.

This is the third procedure from Neil Exley. Neil writes: "The first parameter is the string that is going to contain the program line, and the second parameter is the line number of the required program line. If the second parameter has a number, but a program line of that number does not exist, then it will use the following program line. The default of the second parameter is the current line number. This procedure has many uses, for example, a program line can be copied into a string (using 'lines'), altered around (using INSTRING), and be put back into the program (by using KEYIN). This is more flexible than the command ALTER as this specifies line numbers.”

In fact, I have used PROC lines in conjunction with another procedure to provide an "auto line-split" - see the next item!

Comments: Neil is using RANDOMIZE only to place x as two bytes in a PEEKable system variable. The string s$ is created to contain machine code which loads a register with the line number (x), calls the ROM's "find a program line" subroutine, and transfers the result to register BC, which is passed back as the result of USR. (LENGTH 0 allows this "code in a string" to be tailed, by supplying its location.) The variable U is made to hold the location of the program line, or the location of the variables area if the line number given was past the end of the program. The EXIT IF checks U vs. the system variable VARS to see if this has happened. Y$ holds a formatted version of the line number which is PEEKed in case we are at the following line because our supplied line number didn't exist. To this is added a piece of memory that contains the text of the line, using the line length information that follows the line number in the program area. The DO-loop starting at line 1030 removes any invisible five-byte numbers from the text. (Have a look at the procedure CLEAN in the later PROC KEDIT item in this issue, which does the same thing by a slightly different method.) LOOP UNTIL 1 means "never LOOP" and is a "trick" to allow the use of EXIT IF instead of GOTO.

```
10   lines a$,1000
20   PRINT a$

1000 DEF PROC lines REF x$,x
        LOCAL s$,y$,i,u
        DEFAULT x=DPEEK(23625)
        LET x=INT (x+.5),x$=""
        DO
        EXIT IF x<1 OR x>9999
1010       RANDOMIZE x
           LET s$=CHR$ 33 +CHR3 (PEEK 23670)+CHR$ PEEK (23671)+
           CHR$ 205+CHR$ 110+CHR3 25+CHR$ 68+CHR$ 77+CHR$ 201
           LET u=USR LENGTH(0,"s$")
        EXIT IF u>=DPEEK(23627)
1020       LET y3=USING$("####",PEEK u*256+PEEK (u+1))
           LET x$=y$+MEMORY$()(u+4 TO u+DPEEK(u+2)+2)
1030       DO
              LET i=INSTRING(1,x$,CHR$ 14)
              IF i<>0 THEN LET x$=x$( TO i-1)+x$(i+6 TO )
1040       LOOP UNTIL NOT i
        LOOP UNTIL 1
     END PROC
```

****

PROC SPLIT - automatic splitting of lines into statements.

(Requires the preceding PROC LINES to work.)

Reader Paul Field sent me a big sheaf of interesting stuff that I don't have space for this issue, amongst which was a routine to split up long program lines and automatically re-enter them with a separate statement on every line. As he says, "How often have you made a line of such length that the cursor speed makes editing a nightmare?" Paul made the target line into a REM statement manually to avoid the invisible five-byte numbers problem, got the line into a string, and used a clever but longish method of splitting the line only at colons that were not inside strings. With this to prompt me, and PROC lines just typed in, I wrote a less sophisticated but shorter procedure that uses the output of PROC lines. It doesn't notice if colons are in quotes, so it won't work on lines containing such colons. I have chosen to have each separate statement put back into the program starting at the point where the line came from, incrementing the line number by 1 each time. There won't always be enough space for this, and you might prefer to assign the variable *lin* a high value so as to place the new lines at the end of the program temporarily, as Paul did.

-Add new lines 30-50 and PROC SPLIT to the lines given in PROC LINES previously. The demonstration splits up the first line of PROC LINES.

```
10   lines a$,1000
20   PRINT a$
30   split a$
40   LIST 1000
50   STOP

100  DEF PROC split a$
110     LET lin=VAL a$(1 TO 4)
120     LET p=5
130     DO
           LET q=INSTRING(p,a$,":")
140     EXIT IF q=0
150        KEYIN USING$("####",lin)+a$(p TO q-1)
           LET p=q+1,lin=lin+1
160     LOOP
170     KEYIN USING$("####",lin)+a$(p TO )
180  END PROC
```

Remember that line numbers are important to the logic of IF and ELSE statements, so the split-up version of a line may not work correctly without manual re-joining in some places.

****
UNLIMITED USER-DEFINED GRAPHICS

If you find it irritating having to alternate between KEYWORDS 0 and KEYWORDS 1 to use UDGs, or if you find that having just 21 UDGs is a limitation, here is a method of transferring UDGs to an array using GET. The individual array strings become the equivalent of user-defined graphics, provided you are using a CSIZE *other than zero*. They can also be PLOTed in any CSIZE.

```
10   DIM a$(21,9)
20   KEYWORDS 0
3O   FOR n=1 TO 21
40      PRINT AT 0,0;CHR$ (143+n)
50      GET a$(n),0,175
60   NEXT n
70   KEYWORDS 1
80   CSIZE 8
90   FOR n=1 TO 21
100     PRINT a$(n);
110  NEXT n
```

The array strings have to be 9 characters long to hold the GET strings. An array a$(210,9) would hold 10 complete UDG sets, with every UDG available at the same time!

****
MOVE BUG IN BB 4.0 OPUS DISCOVERY VERSION

This was reported by several users. If you have the Discovery version of BB 4.0, check that PEEK 61889 and 61890 are 251 and 224 respectively; if not, POKE them with those values to correct a serious bug in the MOVE command.

****

PROC KEDIT - EDITING DEF KEYs

This contribution is from Alexander Stols of Molenhoek, The Netherlands. It edits user-defined keys and is much easier to use than the method explained in the manual and in Newsletter no. 6. Alexander writes:

"Variable p points to the first address above RAMTOP, containing either the code for a user-defined key, a window number +128, or 0 (end of definitions). Variable s holds the code for the key you wish to edit, converted to lower case. I have included SHIFT$ 6 to prevent the procedure from finding WINDOW definitions when using wrong input. The DO-LOOP searches through the definitions area, and when the search has been completed, p is zero when the key has not been defined, or otherwise it holds the address where the definition was found. If there is one, it is picked up. Procedure CLEAN deletes invisible five-byte forms inside the string, if there are any, and then you can do the actual editing. I used EDIT LINE, because it makes entering keywords easier, at least if you use LIST FORMAT 1 or 2. After editing the string, it is KEYed IN if you had given a valid key name, giving report C, Nonsense in BASIC, if you had not."

```
10   DEF KEY "1"
     PRINT "asdfg"
20   kedit "1"

9000 DEF PROC KEDIT s$
9010    LOCAL p,s,e$
9020    LET p=DPEEK(23730)+1,s=CODE SHIFT$(2,SHIFT$(6,s$))
9030    DO WHILE PEEK p
9040    EXIT IF s=PEEK p
9050       LET p=p+3+DPEEK(p+l)
9060    LOOP
9070    IF PEEK p THEN LET e$=MEMORY$()(p+3 TO p+2+DPEEK(p+1))
           CLEAN e$
        ELSE LET e$=””
9080    EDIT LINE e$
9090    KEYIN "DEF KEY"""+CHR$ s+""":"+e$
9100 END PROC

9110 DEF PROC CLEAN REF a$
9120    LOCAL t
9130    LET t=1
9140    DO
9150       LET t=INSTRING(t,a$,CHR$ 14)
9160       IF t THEN DELETE a$(t TO t+5)
9170    LOOP WHILE t
9180 END PROC
```

****

BETA BASIC 3.0 AND 4.0 MANUALS IN DANISH

Subscriber Leif Mortensen (Bryggervangen 29, DK-7120 Vejle 0, Denmark) has translated both manuals into Danish, and very well printed they are too. He charges 2 Microdrive cartridges (strange currency!) for a copy of BOTH manuals, which must be a good deal if you speak Danish.

****
TEXT JUSTIFICATION

Back in issue 3 we had a left-justification procedure, PROC prtstr. Recently I was sent two procedures to perform justification on both margins, one by Geoff Stilwell, author of PROC prtstr, and one by Mike Eida, who runs a motoring school in Crawley, W. Sussex. I include Mike's version here because it is a bit shorter, and besides, it's his turn! (Was it Andy Warhol who said that one day, everyone would be famous for 15 minutes? Well Mike, you've had it now...)

```
10   LET q$="This is a test of this procedure. Add more text
     of your own! "
20   LET q$=q$+q$+q3+q$
30   e

1000 DEF PROC e
       LOCAL x$,a,b,c,e,n,s,t
       LET a=1,b=PEEK 57391,n=a,s=0,t=b+1
       DO
          LET x$=q$(n TO t)
          DO UNTIL q$(t)=" "
             LET t=t-1
          LOOP
          LET x$=q$(n TO t-1)
          DO
          EXIT IF LEN x$=b
             DO
                LET a=INSTRING(a,x$," ")
             EXIT IF a
                LET a=1
             LOOP
             LET x$=x$( TO a)+" "+x$(a+1 TO),a=a+s+2,c=a>b,e=a
             <=b,a=(1 AND c)+(a AND e),s=(s+1 AND c)+(s AND e)
          LOOP UNTIL LEN x$=b
          PRINT x$
          LET a=1,n=t+1,t=t+b+1
       LOOP UNTIL t-1>=LEN q$
       PRINT q$(n TO )
     END PROC
```

This was on disc so I didn't have to type it in. You'd better add a few line numbers - line 1000 is too big to edit easily. As written, the text has to be in q$, but of course you could pass it as a parameter instead. The procedure works well in various CSIZEs, using a PEEK to determine how many characters per line to use. (Apart from that, I don't know much about how it works - that huge LET statement looks tough!) To use it with a printer, you could alter the two PRINT statements to print to e.g. stream *str* and assign *str* to be either 2 for the screen or 3 for the printer. Alternatively, just OPEN 2;"t" (or perhaps "p") to divert screen output to the printer. CSIZE will still determine the characters per line. The printing speed of my dot-matrix printer was hardly slowed by the justification operation. Mike suggested that I could tidy the Newsletter (!!) but I must admit I prefer "ragged right" normally. I find lines with a few long words can look pretty odd with right-justification. It is a pity that you normally have to use a whole number of spaces for "padding" lines. (The Beta Basic 3.0 manual was printed on a daisy-wheel printer using a special program I wrote for the job. It used fractional spaces and variable spacing of the letters within words, which combined with proportional spacing, gave a very even appearance. Alas, I don't have the time for daisywheel printing anymore!)

****

PROC VKILL - DELETING VARIABLES

Reader Mike Eida (Crawley, W. Sussex) asked me for a way of disposing of unwanted numeric variables without losing any long strings or arrays that contain valuable data. At the moment, DELETE a$ can be used to get rid of an unwanted string or string array called a$, and DELETE a() will get rid of a numeric array a(). Simple numeric variables do not take up as much memory individually as strings and arrays, but there tend to be many more of them. I have written a PROC VKILL to remove a named variable; lines 10 to 70 demonstrate it. All types of variable are dealt with.

The procedure has quite a few sophisticated (well, actually, downright obscure!) features, which I will try to explain. Notice that instead of a normal parameter being expected, the DEF PROC specifies DATA. This means that the pointer used by READ will be set to just after the name of the procedure when you use it by typing VKILL. If you type e_g. VKILL "abc " then the next READ will use "abc" as a DATA statement. READ would give x$ the value "abc". Later on, we could use this to get the PROC to delete the numeric variable abc. However, we can do things more elegantly by using READ LINE x$ instead of READ. This allows us to avoid using the quote marks so we can type e.g. VKILL abc to get x$="abc". Notice we are doing something quite different from normal parameter passing; instead of abc being evaluated as a number and the value being used by the procedure, we are telling the procedure the name of a variable we want removed.

Having got the name, the procedure decides if it is a string or string array name (by looking for any character followed by "$") or a numeric array name (by looking for any character followed by "()"). If so, DELETE can be used at once to get rid of it. KEYIN does the equivalent or typing DELETE, then the variable name, then "()" if needed.

There remain three variable types: FOR-NEXT loop variables, normal single-letter numerics, and multi-letter numerics. Each is stored somewhat differently in the variables area. If we can work out the stored form, we can search the variables area for it, alter the variable to a string using a POKE, and use the DELETE command to get rid of it.

We cannot tell the difference between N the FOR-NEXT variable and N the single-letter normal numeric variable; however, we can force a single-letter to *become* a FOR-NEXT variable by using it in a FOR-NEXT loop (line 1060). This also changes its value to a known quantity (zero), making a search for the variable plus its value possible. (This removes possible ambiguity.) For multi-letter numerics, LET does the same job. The character values of the name are altered to code for type, matching the system used in the variables area. SIZE is assigned with a value related to the length we need to delete from the variables area. It is used in y$ which is what we will POKE the number with when we find it, transforming the number into a string we can DELETE. (We definitely *will* find the number, since line 1060 will create it if it doesn't already exist.) The fairly horrific looking line 1080 does the actual search of the variables area for the numeric variable name, followed by a value of zero (stored as 5 CHR$ 0's). FOR-NEXT variables have 13 extra bytes after this, but we don't need to search for them specifically. After the POKE in line 1090, that part of the variables area looks as though z$ is stored there, and DELETE is used.

(It doesn't matter if z$ is in use by the main program, since it is LOCAL. This also means, unfortunately, that you cannot VKILL size, posn, x$, y$ or z$.)

The ambitious among you might like to try modifying the procedure to VKILL a whole list of variables, or a given range, such as A to Z.

```
10   FOR a=1 TO 2
     NEXT a
20   LET b=123
30   LET c$="asdfg"
40   DIM d(10)
50   LIST DATA
60   vkill a
     vkill b
     vkill c$
     vkill d()
70   LIST DATA

1000 DEF PROC vkill DATA
1010    LOCAL posn,size,x$,y$,z$
1020    READ LINE x$
1030    IF LEN x$=2 THEN
           IF x$(2)="$" THEN
              KEYIN "delete "+x$+"()"
              GO TO 1110
1040    IF LEN x$=3 THEN
           IF x$(2 TO 3)="()" THEN
              KEYIN "delete "+x$
              GO TO 1110
1050    LET x$=SHIFT$(2,x$)
1060    IF LEN x$=1 THEN
           LET size=16
           KEYIN "for "+x$+"=0 to 0"
           LET x$=CHR$ (CODE x$+128)
        ELSE
           LET size=LEN x$+2
           KEYIN "let "+x$+"=0"
           LET x$(1)=CHR$ (CODE x$(1)+64)
           LET x$(LEN x$)=CHR$ (CODE x$(LEN x$)+128)
1070    LET y$=CHR$ 90+CHR$ size+CHR$ 0
1080    LET posn=INSTRING(DPEEK(23627),MEMORY$()( TO DPEEK(236
        41)),x$+CHR$ 0+CHR$ 0+CHR$ 0+CHR$ 0+CHR$ 0)
1090    POKE posn,y$
1100    DELETE z$
1110 END PROC
```

****

THE VARIABLE "RT" IN BB'S SAVE/LOAD ROUTINE

A few users have had trouble with Beta Basic's habit of changing RAMTOP (the top limit of memory usable by BASIC). This occurs when you define a DEF KEY or a WINDOW, or when you use CLEAR with a value. BB'S SAVE routine in line 1 of the Basic loader looks at the relevant system variable and saves from RAMTOP to the end of BB'S CODE, which is just before the user-defined graphics area. Any WINDOWS and DEF KEYS are therefore saved with the program. The variable RT records the value of RAMTOP, and it is also saved, allowing the Basic loader in line 2 to set up the system variable correctly when you come to reload that copy of Beta Basic later on. You should never use a statement like CLEAR 46000 when loading BB'S CODE, unless you are very sure what you are doing, since the required space is variable and likely to grow as bugs are reported and fixed.

****
TRACE ROUTINE

This contribution is from Arthur Yarnell (Warley, W. Midlands). He writes that he started computing with a ZX81 kit and is now a system manager on a MicroVax. He still uses Beta Basic and the Spectrum, though! The routine splits the screen into two windows, program output going to the right-hand side, and the text of the line being executed appearing on the left. The current line and statement numbers are displayed too. I have hacked the program about a bit to remove value-checking which was performed by an additional procedure - I hope without adverse effect. Lines 200 onward are just some junk I put in as a test. RUN, or GO TO 5, and enter 200 as a start line and then either 0 for single stepping, or e.g. 50 for slowed execution.

```
5    TRACE 0
10   INPUT "start line ";r
20   INPUT "Pause length ";wait
30   PRINT "Use the SPACE BAR to step"'" through the programme
     "'"Type WINDOW 0 when finished to"'"return the screen to
       normal."'"Press ENTER to continue."
40   GET i$
     WINDOW 1,0,175,128,176
     WINDOW 2,128,175,128,176
50   WINDOW 1
     CSIZE 4,8
     PAPER 7
     INK 0
     CLS
60   WINDOW 2
     CSIZE 4,8
     PAPER 0
     INK 7
     CLS
     TRACE 70
     GO TO r
70   DEFAULT lnn=0
     WINDOW 1
     IF lnn<>lino THEN
       CLS
       LIST lino-1 TO lino
80   PRINT AT 21,0;"Line=";lino,"Stat=";stat
     WINDOW 2
90   PAUSE wait
     LET Inn=lino
     RETURN
200  PRINT "two hundred"
210  PRINT "here is 210"
220  PRINT "stat 1"
     PRINT "stat 2"
     PRINT "THREE! "
```

****
DISCOVERY USERS CLUBS

I have received 10 issues of the Dutch Discovery Users Club magazine, which is very interesting. The issues were translated into English, and although the English edition will cease to be available soon (with about issue 13, I think) they are available as back issues. There is a lot in them to interest the Discovery user. Early issues have about 15 A4 pages, later ones 32 A5 pages, full of programming articles, product reviews, hardware info., etc. etc. The style is less formal than mine, with literal translation from the Dutch adding an interesting extra flavour. (Like "the rumours weren't just sucked out of some big thumb."!) The magazine is run by rather more people than this one - they have a help-Line, a program bank supplying discs full of cheap software, and working groups on modems, RTTY/slow scan TV, teaching, hardware and genealogy! The address to use is:

DICK KRUITHOF, BOEIERKADE 6, 2725 CH ZOETERMEER, NETHERLANDS

Payment in advance to their Post Office account in Holland is required: ac. No 5241782 c/o R.O. Alders, Utrecht. (Or you can pay in cash.) Issues 1 to 5 have been combined at a cost of £5.33. Issues 6 to 10 cost £2.50 each. There is also a U.K. based Spectrum Discovery Club, with a Newsletter that I haven't seen yet, and programs on disc - everything very cheap. For details, send an S.A.E. to:

PETER LILLEY, 8 RAINHAM CRES., BLACK HILL,
KEIGHLEY, W. YORKS, BD21 2TP.

The 2.2 ROM needed to use the Discovery with 128K Spectrums (and which works with 48K machines too) can be obtained for £10.00 from its author:

DAVE CORNEY, 6 HAMPTON ROAD, COTHAM, BRISTOL, BS6 6HJ

(You can check your ROM version by typing: PRINT USR 8.)

****

SPECIAL OFFER - 10% OFF THE "DUMPY 3" SCREEN COPIER PROGRAM.

I came across this very versatile screen dump program a few months ago. It is very easy to use - by replying to a few questions on the screen, you can create anything from a 100-byte routine to do a simple dump, to a longer routine to do a large size shaded, sideways dump. Over 1000 different styles are possible. The routines can copy any specified piece of the screen. All the routines can be located wherever you like in memory, which is very useful when combining them with Beta Basic or other large programs. At least 14 different parallel and serial interfaces are supported; the printer needs to be EPSON compatible. Finally, the manual is good and Bradway Software offer very good customer support. The program normally costs £8.50 (anywhere in Europe) with P&P included. At 10% off that will be £7.65. You can buy the program from:

BRADWAY SOFTWARE, 33 CONALAN AVENUE
SHEFFIELD S17 4PG, ENGLAND

Just mention the BB Newsletter to get the discount.

****

READERS' LETTERS

Dear Andy,

Andy Hollis (Newsletter 7) wants to "kill line 0 prior to saving" and you have told him just how this can be done. I think, however, that his real aim is better served by:
'SAVE 1 TO ;"name"', for this produces the same taped program, but leaves BB intact for further use.

Ettrick Thomson, Aldeburgh, Suffolk

Dear Andy,

In the manual it states that it is possible to POKE locations 56866 and 56870 to change the CLOCK command into a stopwatch. What values should I POKE into these locations to achieve this? Is there a POKE (or POKEs) that will allow the use of procedure names that do not begin with a letter?

Robert Dickson, Blackheath, London.

Location 56866 holds 1/50ths. of a second per clock advance. Currently it is 50 - i.e. 1 second. Make it 5 for one advance per 5/50ths. (1/10th.) of a second. POKE 56870 with 58; this is the value of the most significant digit of "seconds" which will cause an "overflow" and make "seconds" go back to zero and "minutes" increase by one. CHR$ 58 is just greater than CHR$ 57 ("9") so "seconds" (actually now 1/10ths. of a second) can increase to 99 before begin reset, instead of 59 as before.

My thought on PROCs starting with a non-letter was "what for?" but I suppose it might be useful for e.g. a PROC called "+" for adding something. You can remove the check for a letter by POKEing locations 62128, 62129, 62130, 53775, 53776 and 53777 with zeros.

Dear Andy,

I have been trying, unsuccessfully, to write a dictionary program using INSTRING to help with crosswords. The idea being to enter a string, with a hash replacing missing letters, and then print a list of all words that fitted that string. Could you ask, through the Newsletter, if any fellow subscriber has done anything similar, or could offer any suggestions as to how it might be done.

Dave Swales, Gainsborough, Lincs.

In the past I did a program conversion for SCRABBLE, which has a dictionary of over 20,000 words. You probably need something that large to be really useful. If you think about word length and available memory, you will realise that you need data compression techniques. For example, you can designate different classes of words - those that can be followed by "S", "ED" and "ING" for example; e.g. JUMP, JUMPS, JUMPED, JUMPING. This saves a lot of space. I wouldn't want to attempt such a project without an "off-the-peg" dictionary, myself - too much typing! Bradway Software are releasing a product of this kind (see elsewhere in this issue for their address).

Dear Sir,

A useful tip when you transfer BBC programs to the Spectrum: Set XRG to 1280 and YRG to 1024, this will allow DRAWing and PLOTing performed by the BBC program to be scaled to fit the Spectrum screen. It should be noted that the BBC clips values outside this range and a program may try to plot outside this area, producing an INTEGER OUT OF RANGE report on the Spectrum.

I enclose a program for a Mouse Pointer. It will only work with a Kempston Mouse though.

Garry Rowland, Dagenham, Essex

I'll send copies of the House program to those interested, on receipt of an S.A.E.

Dear Dr. Wright,

I have a random-access file on disc... sometimes a window does not take on the preset attributes, and the input variable, when sent to the data file, is preceded by the colour codes. It can be overcome with a blank PRINT statement...

Paul Field, Halsham, N. Humbs.

The INK, PAPER, OVER, FLASH and BRIGHT statements actually PRINT the appropriate control codes to the current stream. Normally, this is stream 2, the one used by a standard PRINT, but if you have just used a disc file, or the printer, then the current stream will be the one associated with the disc, or the printer. So something like: LPRINT "testing": INK 2: PRINT "Not red ink!" will send the control codes to the printer, not the screen. A dummy "PRINT;" before the "INK 2" will stop this, as you say. The ROM author should have had the colour commands set the current stream to 2 before printing their control codes.

Dear Andy,

Do you use Beta Basic? If so, what programs do you have? I have written a Data Base, Font Editor, and a UDG editor.

Stephen Folly, Harmondsworth, Middx.

Is the Pope a Catholic? Do bears... Hell, of course I use Beta Basic! Most often, for playing with Newsletter contributions, but often I use it for short programs that help with assembly language programming. I use SORT and INARRAY a lot for record keeping, but in a messy program knocked together in a hurry. I must rewrite it someday... why not send in your Database? But the program I use most often (since I am really an assembly language programmer) is the Assembler / Disassembler /Monitor etc. package PYRADEV on the Amstrad CPC. It is so much better than the one I used to use on the Spectrum that now I use it for everything. I send assembled machine code to the Spectrum via the 128's built-in RS232 port. When I write for the PCW, the CPC disc can just be transferred.

Dear Andy,

I have been having some difficulty with the REF Command. For example, when I try REF "file" the first reference i.e.
SAVE *1;"file" is correctly found. However subsequent references are not found. It appears that when a Discovery type command is found, Beta Basic forgets to keep looking. It is particularly annoying when developing Basic programs which must later run without Beta Basic.

Gerry Fisher, Duffield, Derbyshire

*The problem is due to control being lost to the Discovery when the non-BB syntax is found. SAVE 1;"file" doesn't cause this problem. The best solution I can offer is to use ALTER to make all SAVEs / LOADs into the BB form temporarily; e.g: ALTER " SAVE "+"1" TO " SAVE "+"*1" (and vice versa) with SAVE being a keyword.*

****
I have moved! Although post will still reach me pretty rapidly via 92 Oxford Road, you can clip a few days off by writing to: BB

NEWSLETTER, 24 WYCHE AVE., KINGS HEATH, BIRMINGHAM B14 6LQ